DØ Run II Data Distributor Design

Gerald M. Guglielmo[*], Carmenita Moore[†]

(FNAL CD/ODS/OSP)

February 25, 1999

Version 0.2

Abstract

This document serves as a basic design for the DØ Run II Data Distributor. The Data Distributor is responsible for receiving events from the collectors/routers and distributing events according to requests made by Examine client programs. The basic requirements[1] specify that an Examine client declares classes of events which the Data Distributor will store to satisfy later requests from the Examine client. For each Examine client, the Data Distributor must store events separately and management of the stored events is initially configurable by the Examine client.

*gug@fnal.gov

†carmenita@fnal.gov

Contents

1 General Overview

The Data Distributor is an online event queuing system for the monitoring of data in parallel to the data being written to tape. Programs which would like to receive events from the online pipeline make a request to the data distributor to allocate a queue and specify which trigger subsets they would like to study. Each program making a request is allocated a separate queue. The data distributor will queue events which match any of the trigger subsets and apply any prescales requested. When a request for an event is seen, the distributor will remove the oldest event in the queue and send it to the requester. Statistics for the queue can also be requested. The system is designed to handle multiple input connections from the trigger pipeline and multiple monitor program connections. The expected nominal input rate from the trigger pipeline is 50 Hz of events with an average size of 250 KB (12.5 MB/sec). The data distributor runs under OSF1, IRIX and Linux.

1.1 Event Queue Manager

The event queue manager serves as the basic interface to the event queues in the Data Distributor. The event queue manager has a processor for handling messages necessary for the management of the event queues. For each type of message, the manager will register a callback function which will take the appropriate actions to process the message. The manager is responsible for maintaining a separate queue for each examine client which is actively connected. Events are placed on the a queue if they have a trigger or stream id that matches one in the list associated with the queue, and they pass the appropriate prescale criteria.

The event queue manager can remove events from the queue if the queue is full and the overwrite flag is set to true. In this case, the manager can remove the event which has been in the queue the longest and place a new event on the queue. If the overwrite flag is not set and the queue is full, the manager will discard the new event.

When a request for an event is received, the event queue manager must determine which examine client made the request and determine which event queue it is maintaining for the client. Once the appropriate queue has been determined, the manager will signal that the outstanding requests for events for that queue should be updated to include the new request. The actual removing and sending of an event is handled by a processor associated with the queue.

When a request for statistics is received, the event queue manager must first determine the type of the request. There are two types of statistics requests. The first type is for statistics on the queue associated with the requesting client and the second type is for all queues currently maintained by the manager. Next the manager will collect the statistics from the appropriate queue or queues and send it to the client.

The connection stop message signals that a client connection has been lost. After determining the appropriate queue for that connection, the manager will flush the queue and then delete and the associated processor.

1.2 Event Queue Statistics Management

Each queue has associated with it additional information so that it can be properly maintained by the event queue manager. This information includes parameters which state the maximum depth of the queue, if events on the queue can are allowed to be over-written, which connection (and thus examine client) the queue is maintained for, the number of events that are currently requested for sending, a list of statistics and parameters for each trigger and stream id accepted by the queue, and a flag to indicate that the queue is still considered active.

The statistics information and id specific parameters (prescales at the moment) are grouped separately for each trigger and stream id accepted by the queue. The statistics information is further grouped as an event total and a event size total for various conditions. Values are accumulated for events seen by queue which match the trigger or stream id, events which are deleted because the queue is full, events deleted because of the global prescale, events over-written because the queue is full, events currently in the queue, and events sent to the client. These statistics are used for calculating prescales and to satisfy requests for statistics from examine clients.

1.3 Message Handling

The distributor uses the DØme client-server package for communications. This package allows for the registering of callbacks to handle each type of message desired. The registering of each callback specifies the processor that the callback is being registered with, the function to be called and the object to associate the callback with. When a message is received by the processor, the function that was registered for that message is called.

1.4 Memory Tracking

The Data Distributor has the ability to track the potential usage of memory by the program. Since the queues maintained by the event queue manager and queues associated with the processors allocate and free memory as needed and can contain messages of vastly differing size at any given time, the amount of memory in use is constantly varying. Therefore, a means of estimating the potential usage of memory has been implemented based on an estimated expected event size. The closer the estimated size is to the maximum allowed message size (maximum event size effectively), the less likely a memory allocation failure due to insufficient memory will occur.

The queue memory potential usage is tracked by a static instance of the tracking class. This tracker initially determines the maximum available memory that can be allocated at runtime, and then keeps track of how much may potentially be in use for event queues. Each time an event queue is requested, the tracker will check to see if sufficient memory will be available before creating the queue. If there is sufficient memory, then the total of memory potentially in use is updated to include the memory for the new queue and the queue is allowed to be created. If there is not sufficient memory the queue will not be created. When a queue is deleted, for example when a connection is lost, then the tracker updates the total potential usage to indicate that more memory is available again.

2 Initialization

At runtime, the Data Distributor will determine the amount of memory that is available for the program to allocate. In addition to keeping track of the size of allocatable memory, the program will need to monitor how much memory could conceivably be used by the currently available event queues at any given time during the running of the program (at initialization time there will be none). It is expected that the number of queues will vary over time, with queues of differing depth being added and removed. Event sizes will differ from event to event based on many factors and so one must assume potentially full queues of maximum event size can be accommodated without generating an error. All of this information will be used to assure that enough memory is available to handle a new queue request before the new queue will be allowed to be created. The request to create a new queue will be rejected and a message indicating why must be returned to the client.

The Data Distributor will also need to start listening for multicasting of events at this time. As an event is received, the Data Distributor will need to determine if it's trigger or stream criteria match any of the requested types for each of the current event queues.

3 Event Received

When an event is received, the Data Distributor will need to check it's trigger and stream criteria against those stored for each event queue. If the event matches a type requested for a queue, then the event will be added to that queue according to the defined prescales and rules specified for that trigger type in that specific queue given the current state of the queue. The appropriate statistics will be updated for that trigger and queue combination.

The event will only be written to the same queue once, so after finding a match in any given queue the Distributor stops searching on that queue and begins searching the next unchecked queue. If found to match the requirements of another queue, then the event will also be added to that queue and the appropriate statistics updated. In this way, a event can be written to zero or more queues, but only once to any given queue. The Data Distributor is not allowed to communicate back to the collectors/routers, it is only allowed to receive messages from these servers.

4 Examine Initial Connection

When an Examine client makes a connection to the Data Distributor, the client will be requesting a queue be created to handle events of the types it would like to receive. The request will contain the depth of the queue, types of events and criteria for handling events of each type. The Data Distributor will use this information to determine if there is sufficient memory available to allow the proper management of this new queue based on the current status of the program. If there is insufficient resources the Data Distributor will refuse to allocate the queue and inform the Examine client the reason for the refusal (insufficient memory in this case). If there is adequate memory resources available, the Data Distributor will create the new queue, update its records on current memory needs and then inform the Examine client that the request has been successful. Events now received will be eligible for placement on the new queue. It is the responsibility of the Examine client to ask the Data Distributor for an event when it is ready to handle it.

Each queue can contain up to a preset maximum number of events, a limit specified by the Examine client for the connection. The queue can store events which match a member in the list of trigger and stream identifications for the queue, be able to prescale separately for each different type of trigger or stream and keep separate statistics for each different type. To handle all of this, each queue should have a means for tracking the information and statistics for each one of the trigger and stream types it will accept. The number of different types is variable and can be as low as one.

5 Event Requested

When a request for an event is received from an Examine client, the Data Distributor will attempt to remove the first event from the queue (the event which has been on the queue the longest). If no events are present, then the program will allow a waiting period for one to appear on the queue (how long?). Perhaps this should be a loop that can periodically check to see if the Examine client has made any other requests while waiting, like a request for status for example. If an event is available, the Data Distributor will remove the event from the queue, update the appropriate statistics for the queue and trigger type, and send the event to the Examine client.

6 Lost Client Connection

When the connection to the Examine client is lost, for whatever reason, the Data Distributor will clean up the event queue associated with that connection by first flushing the events in the queue and then by removing the queue itself. The memory management system will also be informed at this time that the memory for that queue is no longer needed.

7 Start of Run Message Received

Not sure what is required here.

8 End of Run Message Received

Not sure what is required here either

9 Request for Status Received

The Data Distributor will keep track of statistics for each queue reflecting total number and size of events seen at each queue for each event type accepted by the queue and information on the fate of those events. For example, an event may be seen by the queue but not added because of a prescale setting for that type of trigger. This information will be available to the Examine client by means of a specific request for status. There should also be a global status request mechanism which will return the information for all of the queues currently allocated.

I am not sure what the mechanism will be for handling a global request for status which is not made by one of the Examine clients. Perhaps this can be handled by the same mechanism that handles the Examine client connections, only instead of creating a queue it sends the global status message and then deletes itself.

10 Guaranteed Message Delivery

Not sure what is required here beyond messages that are held until they can be successfully sent. Who will manage these messages and how and when they will be sent is unclear.

11 Error Handling

Again, not sure what needs to be done here. The requirements mention generating alarms but does not specify when or what this really means.

12 Distributor Core Classes

The section describes the core classes of the distributor which govern the internal functionality used by the distributor main program. These classes are involved in the proper queueing of events and maintenance of the queuing system. The distributor classes are in the dlg namespace.

12.1 QueueMemory

This set of classes allows potential memory usage to be tracked by the program and also can determine the maximum allocatable memory at runtime. The MemoryChunk class is just a block 5,000,000 characters that is used as a estimated average maximum amount of memory used by each event in a queue. The main class is QueueMemory.

12.1.1 Private Data Members

Values are stored in megabytes.

int _current_queue_mb_memory Estimate of potential memory usage by currently allocated queues and any other functionality specifically registered to the instance.

int _max_memory_mb_per_queue_entry Estimate of maximum potential size of an entry on the queue (not necessarily the physical maximum if lower estimate is practical).

int _max_mb_memory Maximum memory available based on unit of maximum queue entry size.

typedef ACE_Guard<ACE_Thread_Mutex> _current_guard A guard for helpping in thread safety.

mutable ACE_Thread_Mutex _current_lock A lock for thread safety.

12.1.2 Private Member Functions

void memoryLimit() Returns maximum allocatable memory in megabytes.

bool canAllocate(int) Determines enough memory is available to handle potential usage for a queue of specified depth.

12.1.3 Public Data Members

There are no public data members for this class.

12.1.4 Public Member Functions

void QueueMemory Class constructor taking no arguments.

bool allocateQueue(int) Requests that potential memory usage be updated if enough memory for a specified queue depth exists. Returns true if sufficient memory.

bool deallocateQueue(int) Updates potential memory usage to indicate that a queue of size specified has been removed.

void dump() Dumps information on instance to standard output.

12.2 Wrap_Info

This class contains the statistics information for a given trigger or stream id that is accepted by the queue. This class publicly inherits from genStream_Info.

12.2.1 Private Data Members

int _stream_id Stream ID value.

int _trigger_id Trigger ID value.

double _prescale_all Global prescale on events for seen by the queue that match the trigger or stream ID.

double _prescale_bfull Prescale applied when determining if events should be over-written in the queue. Calculated based on current number of events deleted and over-written because the buffer was full for the trigger or stream ID of the new event.

unsigned int _ev_count Number of events seen at the queue matching the trigger or stream ID.

double _kb_count Number of KBytes seen at the queue for events matching the trigger or stream ID.

unsigned int _ev_del_bfull Number of events deleted that match the trigger or stream ID.

double _kb_del_bfull Number of KBytes deleted for events that match the trigger or stream ID.

unsigned int _ev_del_prescale Number of events deleted because of prescale that match the trigger or stream ID.

double _kb_del_prescale Number of KBytes deleted because of prescale for events that match the trigger or stream ID.

unsigned int _ev_ovrt_bfull Number of events over-written because of the buffer being full that match the trigger or stream ID.

double _kb_ovrt_bfull Number of KBytes over-written because of the buffer being full for events that match the trigger or stream ID.

unsigned int _ev_depth Number of events in the queue matching the trigger or stream ID.

double _kb_depth Number of KBytes in the queue for events matching the trigger or stream ID.

unsigned int _ev_sent Number of events sent to the client from the queue matching the trigger or stream ID.

double _kb_sent Number of KBytes seen sent to the client from the queue for events matching the trigger or stream ID.

12.2.2 Private Member Functions

There are no private member functions for this class.

12.2.3 Public Data Members

There are no public data members for this class.

12.2.4 Public Member Functions

Wrap_Info(int,int,double,double) Class constructor taking four parameters. The first and second parameters are the stream and trigger ID values. The third and fourth are the global and buffer full prescale values.

virtual int stream_id() const Returns stream ID.

virtual unsigned int ev_count() const Returns number of events seen at the queue.

virtual double kb_count() const Returns number of KBytes seen at the queue.

virtual void dump() const Dumps information on class to standard output.

int trigger_id() const Returns trigger ID.

unsigned int ev_pass_prescale() const Returns number of events that passed the global prescale.

unsigned int ev_del_bfull() const Returns number of events deleted because the buffer was full.

unsigned int ev_ovrt_bfull() const Returns number of events over-written because the buffer was full.

void new_event(double) Increment statistics for events seen at the queue. Takes one parameter which gives the size in KBytes of the event.

virtual void add_event(double) Increment statistics on events in the queue. Takes one parameter which gives the size in KBytes of the event.

void remove_event(double) Update statistics on events in the queue. Takes one parameter which gives the size in KBytes of the event.

void ovrt_event(double) Update statistics for events over-written. Takes one parameter which gives the size in KBytes of the event.

void delete_event_bfull(double) Increment statistics on events deleted because the buffer was full. Takes one parameter which gives the size in KBytes of the event.

void delete_event_prescale(double) Increment statistics on events deleted because of the global prescale. Takes one parameter which gives the size in KBytes of the event.

void send_event(double) Increment statistics on events sent to client. Takes one parameter which gives the size in KBytes of the event.

double prescale_all() Returns global prescale for the trigger or stream ID.

double prescale_bfull() Returns the buffer full prescale.

DistTrigStats get_statistics() const Returns statistics for a trigger ID in a container for easy access.

12.3 Wrap_Man

This class is the queue manager which allows the over-writing of events in the queue and prescaling of events before writing to the queue. The class inherits publicly from genStream_Man.

12.3.1 Private Data Members

QueueMemory* _ptr_programMemory A pointer to the potential memory usage tracker.

12.3.2 Private Member Functions

virtual genStream* get_streamer(int) Obsolete?

int receive_queue_request(Auto_DistQueReq_message) Callback for handling a queue allocation message.

int receive_sendEvent_request(Auto_sendEvent_message) Callback for handling a send event message.

int receive_sendStatistics_request(Auto_sendStatistics_message) Callback for handling a send statistics message.

int receive_Connection_Stop(Ref_Ptr<Connection_Stop_Msg>) Callback for handling a connection stop message that is generated when a connection is lost.

12.3.3 Public Data Members

There are no public data members for the class.

12.3.4 Public Member Functions

Wrap_Man(QueueMemory*) Class constructor taking one parameter which is a pointer to the potential memory usage tracker.

int queue_request(D0me::connection*,DistQueReq&) Handles the request for a queue to be added without using the callback mechanism. The first argument is a pointer to a connection object for the client and the second argument specifies the parameters for the queue. This function is called by the callback for requesting a queue.

12.4 Wrap_Thread

This class is a container for the event queue and also holds information for the management of the queue including where the statistics information can be found. This class inherits publicly from genStream and Thread.

12.4.1 Private Data Members

typedef Queue<Auto_Event_Message> Inner_Queue A typedef for the type of queue which will be used.

typedef map<int,Wrap_Info*> Info_map A typedef for the map which will provide access to the objects holding the statistics information.

Inner_Queue _cevq The event queue.

int _maxQueueDepth The maximum depth of the event queue.

bool _ovrt_flag The over-write flag for the queue. True means events can be over-written.

D0me::connection* _sendConnectonPtr pointer to the client connection.

int _sendEventsRequested Number of events request to be sent that have not yet been removed from the queue.

bool _queueDesired This queue is still considered active indicating the connection to the client is still present.

Info_Map _info The map that allows access to the objects containing the statistics information.

typedef ACE_Guard<ACE_Thread_Mutex> _Queue_guard

mutable ACE_Thread_Mutex _queue_lock Extra lock to allow event over-writing in the queue.

mutable ACE_Thread_Mutex _queue_clean_lock Extra lock to allow proper ordering in threads when cleanup is to be done.

ACE_Thread_Mutex _access_control Used as part of the condition checking to properly synchronize threads in this class when a cleanup is requested.

ACE_Condition<ACE_Thread_Mutex> _queueControl The condition class used for synchronization when a cleanup of the queue is requested.

12.4.2 Private Member Functions

void* run() Processor for threaded queue.

bool removeEvent(Auto_Event_Message&) Remove event from the queue.

virtual bool check_data_file(double) Obsolete.

virtual bool move_data_file() Obsolete.

bool prescale(unsigned int,unsigned int,double) Returns true if event passes prescale. The first argument is the total number of events including the new one. The second argument is the number of events that have passed. The third argument is the prescale value.

void add_trigger(DistTrigReq&,bool) Adds a trigger to the list of allowed triggers for the queue. The first argument contains the parameters related to the trigger request and the second is the over-write flag.

12.4.3 Protected Member functions

virtual bool write_event(Event_Message*) Retrieves and sends an event to the client.

12.4.4 Public Data Members

There are no public data members in this class.

12.4.5 Public Member Functions

Wrap_Thread(int,bool,D0me::connection*,DistQueReq) Class constructor taking four arguments. The first argument is the maximum queue depth, the second argument is the over-write flag, the third argument is a pointer to the client connection and the fourth argument is the request which contains all the information for each trigger and stream ID acceptable by the queue.

~Wrap_Thread() Class Destructor.

virtual add_event(Auto_Event_Message) Adds an event to the queue if it passes the prescales.

virtual void flush() Flush the events from the queue.

virtual void cleanup() Cleanup the queue and the thread for the queue.

virtual bool active() Return the value indicating if the queue is still active.

virtual int stream_id(int) Returns stream ID for the index passed in.

virtual const genStream_Info& stream_info(int) const Returns class containing statistics information based on index passed in.

virtual void dump() const Dumps statistics to standard output for all trigger and stream ID values allowed by this queue.

virtual bool match(int) Compare trigger type input to the allowed values for the queue and see if one is matched.

virtual void setSendEventRequested() Increment the number of requested events and signal to the processor thread that a new event request has been seen.

virtual D0me::connection* connection() Return connection pointer to client.

DistQueStats get_statistics() const Returns a container class containing statistics information for all the trigger and stream ID values accepted by the queue.

12.5 genStream

An abstract class for holding a queue and information necessary for management of the queue.

12.5.1 Private Data Members

There are no private data members in this class.

12.5.2 Private Member Functions

virtual bool check_data_file(double)=0

virtual bool move_data_file()=0

12.5.3 Protected Member Functions

virtual bool write_event(Event_Message*)=0

12.5.4 Public Data Members

There are no public data members in this class.

12.5.5 Public Member Functions

virtual add_event(Auto_Event_Message)=0

virtual void flush()=0

virtual void cleanup()=0

virtual bool active()=0

virtual int stream_id(int)=0

virtual const genStream_Info& stream_info(int) const =0

virtual void dump() const=0

virtual bool match(int)=0

virtual void setSendEventRequested()=0

virtual D0me::connection* connection()=0

12.6 genStream_Info

Abstract class for statistics information for a trigger or stream ID in a queue.

12.6.1 Private Data Members

There are no private data members in this class.

12.6.2 Private Member Functions

There are no private member functions for this class.

12.6.3 Public Data Members

There are no public data members in this class.

12.6.4 Public Member Functions

virtual int stream_id() const=0

virtual unsigned int ev_count() const=0

virtual double kb_count() const=0

virtual void add_event(double)=0

virtual void dump() const=0

12.7 genStream_Man

This is a general class for a queue managing class. This class inherits publicly from D0me::Processor.

12.7.1 Private Data Members

There are no private data members in this class.

12.7.2 Private Member Functions

int receive_event(Auto_Event_Message) Callback for handling event message.

int receive_status(D0me::Ref_Ptr<D0me::Status_Message>) Callback for handling status message.

bool check_stream_id(int) Checks if stream ID is being managed. Obsolete.

virtual genStream* get_streamer(int)=0

12.7.3 Protected Member Data

typedef map<int,genStream*> _Stream_Map Typedef for a map to hold queue container class instances for management.

_Stream_Map _sm The map allowing access to the queue container instances being managed.

12.7.4 Public Data Members

There are no public data members in this class.

12.7.5 Public Member Functions

genStream_Man() Class constructor taking no arguments.

~genStream_Man() Class destructor.

void dump() const Dump statistics information for queues managed to standard output.

int queue_request(D0me::Connection* queue_id, DistQueReq&) Function to add a queue without going through the callback mechanism.

13 Message ID values

The message ID values for the Distributor are contained in one include file to simplify the expansion for new ID values. All of the message ID values are defined as offsets from a base message ID value.

13.1 Dist_Message_ID

The following message ID values are currently implemented for the Distributor.

const int Dist_Base_Message_ID = 10000;

const int DistQueReq_Message_ID = Dist_Base_Message_ID + 1;

const int sendEvent_Message_ID = Dist_Base_Message_ID + 2;

const int DistFullStats_Message_ID = Dist_Base_Message_ID + 3;

const int sendStatistics_Message_ID = Dist_Base_Message_ID + 4;

14 Testing and Options Classes

This section describes classes that are used for testing and for providing command line options functionality.

14.1 Dump2

This class allows part of a binary memory allocation to be dumped in hex format to standard output. This can be useful in testing the integrity of messages which contain binary data.

14.1.1 Private Data Members

size_t _length Size of data to be dumped in hex format.

size_t _size Size of data and extra formatting information for dumping in hex format.

char* _buf Buffer for storing formatted information.

int _offset An offset value for positioning where in the binary data to begin the dump from.

int _line_size The number of characters to dump per line.

14.1.2 Private Member Functions

void init(size_t) Initial buffer for formatted data.

size_t l2s(size_t) Convert from size of binary data to size for formatted data for dumping.

friend ostream& operator<<(ostream&,Dump2&) Overloaded operator for printing.

14.1.3 Public Data Members

No public data members in this class.

14.1.4 Public Member Functions

Dump2(size_t length=16) Class constructor taking up to one argument for length.

Dump2(const unsigned char* const,size_t length=16) Class constructor taking pointer to binary data and up to one more argument for length.

~Dump2() Class Destructor.

const char* str() const Returns the buffer of formatted data.

const char* str(const unsigned char* const,size_t length=0) Initializes buffer with formatted data for dumpping and returns the buffer.

int line_size() Returns the line size.

Dump2& line_size(int) Returns the instance with new value for the line size.

14.2 Optionsdd

This class provides the functionality for command line options.

14.2.1 Private Data Members

u_short port A port number for listening.

std::string _data_dir This is obsolete

int _verbose Flag for printing extra status information.

int _debug A debug level for the **DØme classes.**

14.2.2 Private Member Functions

There are no private member functions for this class.

14.2.3 Public Data Members

There are no public data members for this class.

14.2.4 Public Member Functions

u_short port() const Returns port number.

const std::string& data_dir() const Obsolete.

int verbose() const Returns value of verbose flag.

int debug() const Returns debug level for **DØme classes.**

15 Distributor Interface Classes

This section describes the various classes that can be used to interface to the Distributor program with an examine client. There are two basic types of interfaces classes available. The first type of interface class is for making requests of the Distributor and the second type is for accessing information received from the Distributor. The information in this section can also be found in the *Interfacing to the DØ Data Distributor* document[2].

15.1 DistFullStats

The DistFullStats class is allows access to information on the statistics for queues being managed by the data distributor. There is also information in the characteristics of the queues. For details on what information is stored, see the section describing the DistQueStats class (section 15.5).

15.1.1 Private Data Members

There are no private data members in this class.

15.1.2 Private Member Functions

There are no private member functions in this class.

15.1.3 Public Data Members

vector<DistQueStats> _queue There is only one public data member in this class called "_queue", which is a vector of DistQueStats objects. Accessing the queue vector is best done through the member functions. The vector starts at index 0.

15.1.4 Public Member Functions

void add_queue(const DistQueStats&) This member function adds information for a queue to the class by adding a DistQueStats object to the vector _queue.

DistFullStats() Class constructor taking no arguments.

DistFullStats(int) Class constructor taking one argument indicating a minimum number of queues that will have information stored in the class.

DistFullStats(DistFullStats&) Class copy constructor.

DistQueStats& getQueStats(int) const This member function takes one argument indicating an index of the _queue vector and returns a DistQueStats object containing information on one queue. The function will not change member data.

int numberQueues() const This member function returns the number of queues that described in the class. The return value indicates the number of entries in the _queue vector.The function will not change member data.

size_t length() const This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

void dump() const This function dumps to standard output information on all queues contained in the vector _queue. The function will not change member data.

15.2 DistFullStats_Message

The DistFullStats_Message class is a DØ client-server (DØme) message class for transmitting over the network statistics information on queues managed by the Distributor. This is a message class that basically wraps a DistFullStats object.

15.2.1 Private Data Members

DistFullStats _fullStats Holds queue statistics information for reporting to client.

15.2.2 Private Member Functions

There are no private member functions in this class.

15.2.3 Public Data Members

The only public data member for this class is the message id value.

static const int MSG_ID The message id for the DistFullStats_Message class.

15.2.4 Public Member Functions

int msg_id() const Returns the MSG_ID for DistFullStats_Message message type. The function will not change member data.

DistFullStats_Message() Class constructor taking no arguments.

DistFullStats_Message(const DistFullStats&) Class constructor taking data object.

size_t length() const This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

DistFullStats& statsRequest() Returns a DistFullStats object containing the queue information that is contained in the message.

size_t in(void*,size_t) This function is used by the client-server framework to decode the message data from XDR format. The first argument is a pointer to the encoded data and the second argument is the size of the data. The return value indicates the size of data decoded and should equal input size if function was successfull.

size_t out(void*,size_t) This function is used by the client-server framework to encode the message data to XDR format. The first argument is a pointer to where the encoded data should be stored and the second argument is the size of the data. The return value indicates the size of data encoded and should equal input size if function was successfull.

void dump() const This function dumps to standard output information contained in the DistFullStats object in the message. The function will not change member data.

15.2.5 Automatic Reference Counting

There is a typedef that will automatically reference count for an object of this message class. This provides a convenient interface for messages of this class.

Auto_DistFullStats_Message is a typedef to use the client-server automatic reference counting.

15.3 DistQueReq

This class specifies the parameters for configuring a queue in the Distributor. The class contains the global queue parameters plus a list of trigger specific parameters for each trigger that the queue will be configured to accept.

15.3.1 Private Data Members

int _maxQueueDepth Maximum depth of queue being requested.

bool _ovrt_flag Flag indicating if queue allows events to be over-written.

vector<DistTrigReq> _triggerRequest A list of trigger requests containing the trigger request information.

15.3.2 Private Member Functions

There are no private member functions for this class.

15.3.3 Public Data Members

There are no public data members for this class.

15.3.4 Public Member Functions

DistQueReq(int,bool,int) Class constructor taking 3 arguments. The first argument specifies the maximum depth for the queue. The second argument indicates whether or not the queue will allow events to be overwritten. The third argument states the minimum number of triggers that will be described in the request class (this option may be removed in the future).

int maxQueueDepth() const Returns the maximum queue depth allowed for the queue being requested. The function will not change member data.

void maxQueueDepthSet(int) The function allows the requested maximum queue depth to be set to a different value in the request.

bool ovrt() const Returns true if the request is for a queue that will allow overwriting of events. The function will not change member data.

void ovrtSet() This function can change the value of the overwrite flag in the request object.

void addtrigger(DistTrigReq&) This functions adds the parameters for a trigger to the queue request.

int numberTrigs() const Returns the number of triggers configured in the queue request. The function will not change member data.

DistTrigReq& getTrigReq(int) const This function returns an object containing the trigger parameters for one trigger type in the request. The argument specifies an index, beginning at 0 and an upper limit less than the number of trigger specified. The function will not change member data.

void removeTrigReqs() This function removes all trigger requests from the the queue request object.

size_t length() const This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

void dump() const This function dumps to standard output information about the queue request. The function will not change member data.

15.4 DistQueReq_Message

15.4.1 Private Data Members

DistQueReq _qrequest Holds the queue request information.

15.4.2 Private Member Functions

There are no private member functions in this class.

15.4.3 Public Data Members

The only public data member for this class is the message id value.

static const int MSG_ID The message id for the DistFullStats_Message class.

15.4.4 Public Member Functions

int msg_id() const Returns the MSG_ID for DistFullStats_Message message type. The function will not change member data.

DistQueReq_Message() Class constructor taking no arguments.

DistQueReq_Message(const DistQueReq&) Class constructor taking data object.

size_t length() const This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

DistQueReq& QueueRequest() Returns a DistQueReq object containing the queue request information that is contained in the message.

size_t in(void*,size_t) This function is used by the client-server framework to decode the message data from XDR format. The first argument is a pointer to the encoded data and the second argument is the size of the data. The return value indicates the size of data decoded and should equal input size if function was successfull.

size_t out(void*,size_t) This function is used by the client-server framework to encode the message data to XDR format. The first argument is a pointer to where the encoded data should be stored and the second argument is the size of the data. The return value indicates the size of data encoded and should equal input size if function was successfull.

void dump() const This function dumps to standard output information contained in the DistQueReq object in the message. The function will not change member data.

15.4.5 Automatic Reference Counting

There is a typedef that will automatically reference count for an object of this message class. This provides a convenient interface for messages of this class.

Auto_DistQueReq_Message is a typedef to use the client-server automatic reference counting.

15.5 DistQueStats

The DistQueStats class contains information on the statistics for a queue reported by the Distributor.

15.5.1 Private Data Members

There are no private data members in this class

15.5.2 Private Member Functions

There are no private member functions in this class.

15.5.3 Public Data Members

int _maxQueueDepth Indicates the maximum number of events that can be stored in the queue.

int _sendEventRequested The number of events requested that have not yet been sent.

bool _ovrt_flag True if the queue allows over-writing of events.

Vector<DistTrigStats> _trig A vector of objects of class DistTrigStats which hold statistics information for a trigger in the queue maintained by the Distributor.

15.5.4 Public Member Functions

DistQueStats() Class constructor taking no arguments.

DistQueStats(int,int,bool,int) Class constructor which takes four arguments. The first argument is the queue depth. The second argument is the number of events requested that have not been sent yet. The third argument specifies whether or not the queue allows the over-writing of events. The fourth argument indicates the minimum number of triggers that will have statistics information added to the object.

DistQueStats(const DistQueStats&) The class copy constructor.

void addTrigger(const DistTrigStats&) This function adds statistics information for a trigger in the queue to the class object.

int numberTrigs() const Returns the number of triggers that are described by the object. The function will not change member data.

size_t length() const This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

void dump() const This function dumps to standard output information contained in the DistQueStats object. The function will not change member data.

15.6 DistTrigReq

The DistTrigReq class is used in a request to the Distributor for specifying the parameters for a specific trigger that will be accepted by the queue being requested.

15.6.1 Private Data Members

int _stream_id Stream ID.

int _trigger_id Trigger ID.

double _prescale_all Global prescale applied to events of this trigger or stream ID when they are seen at the queue.

double _prescale_bfull Buffer full prescale applied if the buffer is full and over-writing of events is allowed. Prescale is based on number of events overwritten and number of events deleted because the buffer is full.

15.6.2 Private Member Functions

There are no private member functions in this class.

15.6.3 Public Data Members

There are no public data members in this class.

15.6.4 Public Member Functions

DistTrigReq(int,int,double,double) Class constructor taking four arguments. The first argument is the stream id accepted by the queue and associated with this trigger. The second argument is the trigger id. The third argument is a prescale value that will be applied to all events matching the trigger (or stream) id. The fourth argument is a prescale that only applies if event over-writing is allowed for the queue. If over-writing is allowed, then the fourth argument specifies a prescale that applies when the queue is full and allows prescaling of the over-writing of events.

int stream_id() const Returns stream id associated with this trigger request. The function will not change member data.

int trigger_id() const Returns trigger id associated with this trigger request. The function will not change member data.

double prescale_all() const Returns prescale value for all events seen by the queue of the given trigger or stream type. The function will not change member data.

double prescale_bfull() const Returns prescale value for when the queue is full. The function will not change member data.

void dump() const This function dumps to standard output information contained in the DistTrigReq object. The function will not change member data.

15.7 DistTrigStats

The DistTrigStats class contains information on the statistics for a trigger request in a queue reported by the Distributor.

15.7.1 Private Data Members

There are no private data members in this class.

15.7.2 Private Member Functions

There are no private member functions in this class.

15.7.3 Public Data Members

int _stream_id The stream id associated with this trigger request being reported.

int _trigger_id The trigger id associated with this trigger request being reported.

double _prescale_all The overall prescale applied to all events seen by the queue that match the trigger or stream id.

double _prescale_bfull The prescale value applied when the buffer is full. The prescale uses the information on number of events accepted and number over-written for the trigger associated trigger or stream type.

unsigned int _ev_count Number of events that match the trigger or stream id that have been seen by the queue.

double _kb_count The total size of events that match the trigger or stream id that have been seen by the queue.

unsigned int _ev_del_bfull Number of events that match the trigger or stream id that have been deleted because the queue was full.

double _kb_del_bfull The total size of events that match the trigger or stream id that have been deleted because the queue was full.

unsigned int _ev_del_prescale Number of events that match the trigger or stream id that have been deleted because of a prescale.

double _kb_del_prescale The total size of events that match the trigger or stream id that have been deleted because of a prescale.

unsigned int _ev_ovrt_bfull Number of events that match the trigger or stream id that have been over-written because the queue was full.

double _kb_ovrt_bfull The total size of events that match the trigger or stream id that have been over-written because the queue was full.

unsigned int _ev_depth Number of events that match the trigger or stream id currently in the queue.

double _kb_depth The total size of events that match the trigger or stream id currently in the queue.

unsigned int _ev_sent Number of events that match the trigger or stream id that have been sent to the client.

double _kb_sent The total size of events that match the trigger or stream id that have been sent to the client.

double _calc_prescale_all The value calculated for the overall prescale based on the statistics of the queue for the trigger or stream id associated with the trigger request.

double _calc_prescale_bfull The value calculated for the prescale applied when the queue is full based on the statistics of the queue for the trigger or stream id associated with the trigger request.

15.7.4 Public Member Functions

DistTrigStats() Class constructor taking no arguments.

void dump() const This function dumps to standard output information contained in the DistTrigStats object. The function will not change member data.

15.8 sendEvent_Message

The sendEvent_Message is a client-sever message class that allows the client to request the Distributor to send events from the queue it is managing for the client. The request allows a variable number of events to be requested at one time.

15.8.1 Private Data Members

int _numberEvents Number of events being requested.

15.8.2 Private Member Functions

There are no private Member functions in this class.

15.8.3 Public Data Members

static const int MSG_ID The message id for the sendEvent_Message class.

15.8.4 Public Member Functions

sendEvent_Message() Class constructor taking no arguments.

sendEvent_Message(const int) Class constructor taking one argument specifying the number of events requested.

int msg_id() const Returns the MSG_ID for DistFullStats_Message message type. The function will not change member data.

size_t length() const This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

int numberEvents() Returns the number of events requested to be sent.

size_t in(void*,size_t) This function is used by the client-server framework to decode the message data from XDR format. The first argument is a pointer to the encoded data and the second argument is the size of the data. The return value indicates the size of data decoded and should equal input size if function was successfull.

size_t out(void*,size_t) This function is used by the client-server framework to encode the message data to XDR format. The first argument is a pointer to where the encoded data should be stored and the second argument is the size of the data. The return value indicates the size of data encoded and should equal input size if function was successfull.

void dump() const This function dumps to standard output information contained in the message object. The function will not change member data.

15.8.5 Automatic Reference Counting

There is a typedef that will automatically reference count for an object of this message class. This provides a convenient interface for messages of this class.

Auto_sendEvent_Message is a typedef to use the client-server automatic reference counting.

15.9 sendStatistics_Message

The sendStatistics_Message class allows a client to request that the Distributor send statistics information for either the queue associated with the connection, or all queues maintained by the Distributor.

15.9.1 Private Data Members

int _requestType Specifies the type of statistics request: 0 for statistics from only the queue associated with the connection; 1 for all queues being managed; greater than 1 is reserved for future use.

15.9.2 Private Member Functions

There are no private Member functions in this class.

15.9.3 Public Data Members

static const int MSG_ID The message id for the sendStatistics_Message class.

15.9.4 Public Member Functions

sendStatistics_Message() Class constructor taking no arguments.

sendStatistics_Message(const int) Class constructor taking one argument which specifies whether only the queue associated with the connection should be reported on (a value of 0) or all queues maintained by the Distributor should be reported on (a value of 1).

int msg_id() const Returns the MSG_ID for DistFullStats_Message message type. The function will not change member data.

size_t length() const This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

int requestType() Returns 0 if the request is only for the queue associated with the connection, or 1 if the request is for all queues.

size_t in(void*,size_t) This function is used by the client-server framework to decode the message data from XDR format. The first argument is a pointer to the encoded data and the second argument is the size of the data. The return value indicates the size of data decoded and should equal input size if function was successfull.

size_t out(void*,size_t) This function is used by the client-server framework to encode the message data to XDR format. The first argument is a pointer to where the encoded data should be stored and the second argument is the size of the data. The return value indicates the size of data encoded and should equal input size if function was successfull.

void dump() const This function dumps to standard output information contained in the message object. The function will not change member data.

15.9.5 Automatic Reference Counting

There is a typedef that will automatically reference count for an object of this message class. This provides a convenient interface for messages of this class.

Auto_sendStatistics_Message is a typedef to use the client-server automatic reference counting.

References

[1] S. Fuess, G. Guglielmo, C. Moore, L. Rasmussen and J. Yu, DØ Run II Data Distributor Requirements (October 22, 1998).

[2] G. Guglielmo, Interfacing to the DØ Data Distributor (January 26, 1999).